A Tighter Insertion-based Approximation of the Crossing Number

Markus Chimani*[1] and Petr Hliněný**[2]

[1] Algorithm Engineering, Friedrich-Schiller-University Jena, Germany
markus.chimani@uni-jena.de
[2] Faculty of Informatics, Masaryk University Brno, Czech Republic
hlineny@fi.muni.cz

Abstract. Let G be a planar graph and F a set of additional edges not yet in G. The *multiple edge insertion* problem (MEI) asks for a drawing of $G+F$ with the minimum number of pairwise edge crossings, such that the subdrawing of G is plane. As an exact solution to MEI is NP-hard for general F, we present the first approximation algorithm for MEI which achieves an additive approximation factor (depending only on the size of F and the maximum degree of G) in the case of connected G. Our algorithm seems to be the first directly implementable one in that realm, too, next to the single edge insertion.
It is also known that an (even approximate) solution to the MEI problem would approximate the crossing number of the *F-almost-planar graph* $G+F$, while computing the crossing number of $G+F$ exactly is NP-hard already when $|F| = 1$. Hence our algorithm induces new, improved approximation bounds for the crossing number problem of F-almost-planar graphs, achieving constant-factor approximation for the large class of such graphs of bounded degrees and bounded size of F.

1 Introduction

The crossing number $\mathrm{cr}(G)$ of a graph G is the minimum number of pairwise edge crossings in a drawing of G in the plane. The crossing number problem has been vividly investigated for over 60 years, and yet only little is known about it. See [23] for an extensive bibliography. While the problem's approximability is still unknown, several approximation algorithms arose for special graph classes.

The best known polynomial algorithm for the crossing number of general graphs with bounded degree [1, 12] approximates, within a factor of $\log^2 |V(G)|$, the quantity $|V(G)| + \mathrm{cr}(G)$, not directly $\mathrm{cr}(G)$. The known constant factor approximations restrict themselves to graphs following one of two paradigms (see also Section 4): they either assume that the graph is embeddable in some higher surface [13, 18, 20], or they are based on the idea that only a small set of graph elements has to be removed from G to make it planar: removing and

* Markus Chimani was funded by a Carl-Zeiss-Foundation juniorprofessorship.
** Petr Hliněný has been supported by Eurocores grant GIG/11/E023 and Czech Science Foundation grant P202/11/0196.

re-inserting them can give strong approximation bounds [3,8,19]. In this paper, we follow the latter idea and first concentrate on the following problem:

Definition 1.1. *Given a planar graph G and a set F of k edges (vertex pairs, in fact) not yet in G. The* multiple edge insertion *problem* MEI (G, F) *is to find the minimum number* $\mathrm{ins}(G, F)$ *of crossings necessary to draw $G + F$ (G including the edges F) such that the subdrawing restricted to G is planar. In other words, we ask for some* planar embedding *of G such that we can* insert *the edges F into this embedding making the least possible number of edge crossings.*

For general k, the MEI problem is known to be NP-hard [24], based on a reduction from *fixed linear crossing number*; for fixed $k > 1$ the problem complexity is open.

The case $k = 1$ of MEI is known as the *(single) edge insertion* problem and can be solved in linear time [17], as we will briefly summarize in Section 2.2. Let e be the edge to insert, then denote the resulting number of crossings by $\mathrm{ins}(G, e)$. Let $\Delta(G)$ denote the maximum degree in G. It was shown [3,19] that $\mathrm{ins}(G, e)$ approximates the crossing number $\mathrm{cr}(G + e)$ —i.e., of the graph containing this edge e—within a multiplicative factor of $\lfloor \frac{1}{2}\Delta(G) \rfloor$ achieved in [3], and this bound is tight. Notice also that already computing $\mathrm{cr}(G + e)$ exactly is NP-hard [4].

Another special case of the MEI problem is when one adds a new node together with its incident edges; this is also polynomially solvable [6] and approximates the crossing number of the resulting *apex graph* [8]. These are the only two types of insertion problems which are currently known to be in P.

Nevertheless, it has been proven in [8] (see Section 4) that a solution (even an approximate one) to MEI (G, F) would directly imply an approximation algorithm for $\mathrm{cr}(G+F)$ with planar G. Just recently, Chuzhoy et al. [10] have shown the first algorithm efficiently computing an approximate solution to the crossing number problem on $G + F$ with a help of a multiple edge insertion solution. Precisely, Chuzhoy et al. [10] achieve a solution with the number of crossings

$$(1) \qquad \mathrm{cr}^{\mathrm{aprx}}(G + F) \;\le\; O\big(\Delta(G)^3 \cdot |F| \cdot \mathrm{cr}(G + F) + \Delta(G)^3 \cdot |F|^2\big)$$

(without giving explicit constants). Though not mentioned explicitly in [10], it seems that their results also give an approximation solution to MEI (G, F) with the same ratio, at least in the case of 3-connected $G+F$. A further approximation result, though not directly related to our topic, was given by Chuzhoy in [9].

In this paper, we present an alternative approach to the one proposed in [10]: We directly give an efficient algorithm approximating a solution of MEI (G, F), and then employ the aforementioned result of [8]. On the one hand, our approach is algorithmically and implementationally simpler, virtually only building on top of well-studied and experimentally evaluated sub-algorithms. On the other hand, it gives stronger approximations, cf. (3), as well as better runtime bounds. Our algorithm, in fact, seems to be the first *directly implementable* algorithm in this area, next to the single edge insertion. We are going to show:

Theorem 1.2. *Given a connected planar graph G and an edge set F, $F \cap E(G) = \emptyset$, Algorithm 3.1 described below finds, in $O(|F| \cdot |V(G)| + |F|^2)$ time, a solution to the* MEI (G, F) *problem with* $\mathrm{ins}^{\mathrm{aprx}}(G, F)$ *crossings such that*

$$\mathrm{ins}^{\mathrm{aprx}}(G,F) \;\leq\; \mathrm{ins}(G,F) + \left(\lfloor \tfrac{1}{2}\Delta(G) \rfloor + \tfrac{1}{2}\right) \cdot \left(|F|^2 - |F|\right). \tag{2}$$

Consequently, this gives an approximate solution to the crossing number problem

$$\mathrm{cr}^{\mathrm{aprx}}(G+F) \leq \lfloor \tfrac{1}{2}\Delta(G) \rfloor \cdot 2|F| \cdot \mathrm{cr}(G+F) + \left(\lfloor \tfrac{1}{2}\Delta(G) \rfloor + \tfrac{1}{2}\right)\left(|F|^2 - |F|\right). \tag{3}$$

Notice the constant-factor approximation ratio when the degree of G and the size of F are bounded. Further consequences of our main result will be discussed below. We, moreover, remark that the assumption of connectivity of G is neccessary in the context of Algorithm 3.1 (if G were not connected, then the approximation guarantee of Algorithm 3.1 for $\mathrm{ins}^{\mathrm{aprx}}(G, F)$ would be just the same as for $\mathrm{cr}^{\mathrm{aprx}}(G + F)$ in line (3)).

2 Preliminaries

2.1 Decomposition trees

We consider multigraphs by default. Our algorithm will use suitable tree-structured decompositions of the given planar graph, according to its connectivity.

Definition 2.1 (BC-tree). *Let G be a connected graph. The* BC-tree $\mathcal{B} = \mathcal{B}(G)$ *of G is a tree that satisfies the following properties:*

i. $\mathcal{B}$ has two different node types: B- and C-nodes.
ii. For every cut vertex in G, $\mathcal{B}$ contains a unique corresponding C-node.
*iii. For every maximal two-connected subgraph (*block*) in G, $\mathcal{B}$ contains a unique corresponding B-node.*
iv. No two B-, and no two C-nodes are adjacent. A B-node is adjacent to a C-node iff the corresponding block contains the corresponding cut vertex.

To further decompose the blocks, we consider SPQR-trees for each non-trivial B-node (i.e., the block containing more than a single edge). This decomposition was first defined in [11], based on prior work of [2,22]. Even though more complicated than the BC-tree, it requires also only linear size and can be constructed in linear time [15,21]. We are mainly interested in the property that an SPQR-tree can be used to efficiently represent and enumerate all (potentially exponentially many) planar embeddings of its underlying graph. For conciseness, we call our tree *SPR-tree*, as we do not require nodes of type Q.

Definition 2.2 (SPR-tree, cf. [5]). *Let H be a biconnected graph with at least three nodes. The* SPR-tree $\mathcal{T} = \mathcal{T}(H)$ *of H is the (unique) smallest tree satisfying the following properties:*

i. Each node ν in $\mathcal{T}$ holds a specific (small) graph $S_\nu = (V_\nu, E_\nu)$, $V_\nu \subseteq V(H)$, called a skeleton. *Each edge f of E_ν is either a* real *edge $f \in E(H)$, or a* virtual *edge $f = \{u, v\}$ where $\{u, v\}$ forms a 2-cut (a* split pair*) in G.*

ii. *$\mathcal{T}$ has only three different node types with the following skeleton structures:*

S: *The skeleton S_ν is a simple cycle—it represents a* serial *component.*

P: *The skeleton S_ν consists of two vertices and multiple edges between them—it represents a* parallel *component.*

R: *The skeleton S_ν is a simple triconnected graph.*

iii. *For every edge $\{\nu, \mu\}$ in $\mathcal{T}$ the following holds: S_ν contains a specific virtual edge e_μ which "represents" S_μ. Vice versa, $e_\nu \in E(S_\mu)$ represents S_ν. Both edges e_μ and e_ν connect the same vertices.*

iv. *The original graph H can be obtained by recursively applying the following operation: For the edge $\{\nu, \mu\} \in E(\mathcal{T})$, let e_μ, e_ν be the virtual edges as in (iii.) connecting the same vertices u, v. A merged graph $(S_\nu \cup S_\mu) - e_\mu - e_\nu$ is obtained by gluing the skeletons together at u, v and removing e_μ, e_ν.*

We remark that SPQR-trees have also been used in the aforementioned [10], though with a different approach. For our purpose, we are particularly interested in the amalgamated version of both above trees, chiefly denoted by *con-tree*:

Definition 2.3 (Con-tree). *Given a connected, planar graph G, let the* con-tree *$\mathcal{C} = \mathcal{C}(G)$ be formed of the BC-tree $\mathcal{B}(G)$ which holds SPR-trees $\mathcal{T}(H)$ for all non-trivial blocks H of G.*

Clearly, the linear-sized con-tree $\mathcal{C}$ can be obtained from G in linear time.

Observe that, for each cut vertex x (of G) incident with a block $H \subseteq G$, the nodes with skeletons containing x induce a subtree $T_{H,x} \subseteq \mathcal{T}(H)$. In further considerations it will be useful to imagine that the C-node of x in $\mathcal{B}(G)$ is adjacent to these corresponding nodes $V(T_{H,x})$ (over all blocks H incident with x) within $\mathcal{C}$. We will loosely call this view of $\mathcal{C}$ the *extended con-tree*, while understanding that it is not really a tree.

2.2 Single edge insertion with variable embedding

As noted before, Gutwenger et al. [17] presented an exact linear-time algorithm to solve the single edge insertion problem. Herein, we will only outline some central concepts of this approach. Consider a planar graph G and let v_1, v_2 be the vertices we want to connect by a new edge.

First consider G embedded such that we can deal with its dual graph G^*. We define an *insertion path* to be a path in G^* connecting a face incident to v_1 with a face incident to v_2. The length of this path is then the number of edge crossings necessary to insert the edge $\{v_1, v_2\}$ into embedded G along this path. For a fixed embedding, we can compute the shortest insertion path via a breath-first search (BFS). Now consider G without a fixed embedding. If v_1, v_2 are in two separate connected components, then we can insert the edge $\{v_1, v_2\}$ without any crossings, by choosing sub-embeddings of these components where v_1, v_2 lie on their respective outer faces.

Hence assume G to be connected and compute (in linear time) its con-tree $\mathcal{C}(G)$. Let L be the unique shortest path in $\mathcal{B}(G)$ from a B-node containing v_1 to a B-node containing v_2. The optimal insertion path for $\{v_1, v_2\}$ in G can be obtained by concatenating the optimal insertion paths within the (non-trivial) blocks on this path L; as we can always "flip" (cf. Definition 3.4, C-nodes) the two incident blocks at the common cut vertex to avoid additional crossings. For a block H represented by a B-node on L, let v_i^H, $i = 1, 2$, denote v_i if $v_i \in V(H)$, or the cut vertex in H closest to v_i otherwise. An *insertion path within* H then connects any face incident to v_1^H with any face incident to v_2^H.

To find the optimal insertion path within such a block $H \subseteq G$, let Q_H be the unique shortest path in $\mathcal{T}(H)$ from a skeleton containing v_1^H to a skeleton containing v_2^H. It was shown [17] that only the embeddings of the skeletons within Q_H matter. Roughly speaking, the algorithm walks along these skeletons and fixes a suitable embedding one after another. Finally, an optimal embedding is found and fixed, and one can use a simple BFS algorithm on the dual graph to insert the edge $\{v_1^H, v_2^H\}$ optimally.

Definition 2.4 (Con-chain). *Considering the previous notation, we call a* con-chain *of the edge* $\{v_1, v_2\}$ *the (unique) path* $Q = Q_G(\{v_1, v_2\})$ *resulting from* L *by expanding each B-node* $b \in V(L)$ *into the path* Q_H *where* H *is the block of* G *represented by* b.

Proposition 2.5. *Let* e_1, e_2 *be two insertion edges to the same graph* G. *Then their con-chains* $Q_G(e_1)$ *and* $Q_G(e_2)$ *are either disjoint, or they intersect (within the extended con-tree of* G *in which they lie) in one subpath.*

3 MEI Approximation Algorithm

We can now describe the main algorithm for our Theorem 1.2 (2). In the following, we will always consider the multiple edge insertion problem MEI (G, F) for a planar graph $G = (V, E)$ and an edge set F, $F \cap E = \emptyset$, with $k := |F| > 1$. Let $\Delta := \Delta(G)$ be the maximum degree in G. We will present an algorithm giving a solution to MEI (G, F) that approximates the optimum $\mathrm{ins}(G, F)$ within an *additive factor* (depending only on k, Δ). Afterwards, in Section 4, we will discuss how to obtain an approximation algorithm for $\mathrm{cr}(G + F)$ based on this result.

On a high level, our algorithm proceeds as follows:

Algorithm 3.1. Computing an approximate solution to the multiple edge insertion problem MEI (G, F) for connected planar G.

(1) Build the con-tree $\mathcal{C} = \mathcal{C}(G)$.
(2) Using $\mathcal{C}$, compute single-edge insertions (including the con-chains $Q_G(e)$ of Definition 2.4) for each edge $e \in F$ independently, and centrally store their *embedding preferences* (Definition 3.4).
(3) Fix an embedding Γ of G by suitably (see Algorithm 3.8) combining the embedding preferences from step (2).
(4) Independently compute insertion paths for each edge $e \in F$ into the fixed embedding Γ.
(5) Realize all the insertion paths computed in the prior step.
(5)a) If some insertion paths cross multiple times, exchange subpaths, such that in the end all inserted edges cross each other at most once.

Observe that, by minimality of the paths computed in step (4), the lengths of the paths cannot change by applying step (5)a. In fact, by suitably breaking ties in the BFS algorithms of step (4), the situation of (5)a will never occur. Furthermore, by simple iterative insertion in step (4), even explicit tie-breaking becomes superfluous, cf. Section 5. Hence, in the following we can always consider the paths obtained in step (4) to be free of multi-crossings between any pair of insertion paths.

By using the aforementioned algorithms for building the decomposition tree and the single edge insertions as black boxes, we can directly perform the steps (1), (2), (4), and (5). We will discuss step (3) in Section 3.2. Yet, we can already informally describe the core idea of why the value $\mathrm{ins}^{\mathrm{Alg}}(G, F)$ of the outcome of Algorithm 3.1 approximates $\mathrm{ins}(G, F)$.

Clearly, $\mathrm{ins}^{\Sigma}(G, F) := \sum_{e \in F} \mathrm{ins}(G, e)$—the sum of the individual insertions without considering interdependencies—is a lower bound for $\mathrm{ins}(G, F)$. Moreover, we can compute $\mathrm{ins}^{\Sigma}(G, F)$ exactly in step (2). Hence to give an approximation guarantee for Algorithm 3.1, it is enough to bound $\mathrm{ins}^{\mathrm{Alg}}(G, F)$ in terms of $\mathrm{ins}^{\Sigma}(G, F)$ and a function of k, Δ.

Let $e \in F$ be an inserted edge with the computed con-chain $Q_G(e)$, and $\nu \in V(Q_G(e))$ be a C-,P-, or R-node of $\mathcal{C}$ along it. An embedding preference of e at ν—actually respecting its neighborhood and generally denoted by a node tuple $\mathfrak{c}_\nu$, as formally explained in Definition 3.5—specifies what the embedding of G should locally "look like at ν" to achieve the (independent) optimum $\mathrm{ins}(G, e)$. Roughly, we call such a pair $(\mathfrak{c}_\nu, e)$ a *dirty pass* if, in the embedding Γ of step (3), the embedding preferences at ν and its neighbors has been fixed incompatibly from those individually chosen by e in the previous step (2), cf. Section 3.1 for details.

Theorem 3.2. *Consider a run of Algorithm 3.1 on G and F, with a particular embedding Γ fixed at step (3). Let $k = |F|$, $\Delta = \Delta(G)$. If r is the total number of dirty passes (over all $e \in F$) determined by this Γ, then the number of crossings in the outcome of the algorithm is*

$$\mathrm{ins}^{\mathrm{Alg}}(G, F) \leq \mathrm{ins}^{\Sigma}(G, F) + \left\lfloor \frac{\Delta}{2} \right\rfloor \cdot r + \binom{k}{2}.$$

A full proof will be given in Section 3.1. Here we outline its core idea: As every node of $\mathcal{C}$ with an embedding preference is associated with a 1- or 2-cut in G, any wrongly fixed preference (a dirty pass) can be "repaired" by rerouting the inserted edge close to this cut, crossing at most $\lfloor \Delta/2 \rfloor$ edges incident with a vertex in the cut. Summing over all dirty passes caused by Γ and taking possible crossings between edges of F into account, we get the bound.

It is a fact that the embedding preferences for Γ can be naturally fixed such that, at every node ν of $\mathcal{C}$, not all con-chains of inserted edges disagree with Γ. Consequently, one can give an easy upper bound on r in terms of k as follows: For every dirty pass $(\mathfrak{c}_\nu, e)$ caused by Γ, there is another $f \in F$ such that ν belongs to the con-chain of f and, for a suitable tuple $\mathfrak{c}'_\nu$, $(\mathfrak{c}'_\nu, f)$ is not dirty. So, in particular, the con-chains $Q_G(e)$ and $Q_G(f)$ are not routed through the completely same neighborhood of ν (informally, they "split/merge" at ν), cf. Lemma 3.10 for the concise statement. Since any two con-chains can split/merge at most twice, a coarse bound of $r = O(k^2)$ on the total number of disagreements with Γ easily follows.

Stated formally, the above arguments lead to the following conclusion—overall stronger than (1) of [10], with full details in Section 3.2 and 3.3:

Theorem 3.3 (Theorem 1.2 (2)). *Algorithm 3.1 computes a solution to the* MEI (G, F) *problem for connected planar G with* $\mathrm{ins}^{\mathrm{Alg}}(G, F)$ *crossings such that*

$$\mathrm{ins}^{\mathrm{Alg}}(G, F) \;\leq\; \mathrm{ins}^{\Sigma}(G, F) + \left(2\left\lfloor\frac{\Delta}{2}\right\rfloor + 1\right)\cdot\binom{k}{2}$$

where $k = |F|$, $\Delta = \Delta(G)$, and (also computed thereby) $\mathrm{ins}^{\Sigma}(G, F) \leq \mathrm{ins}(G, F)$ *is a lower bound on the optimal solution.*

3.1 Embedding preferences and estimating additional crossings via dirty passes

In order to discuss how to obtain an embedding Γ suitable for all edge insertions, we first have to concisely define *embedding preferences* that record the desired local embeddings of G from each independent single edge insertion in step (2).

Definition 3.4 (Embedding preference). *Consider a single edge insertion of an edge $e \in F$ into G. As argued in Section 2.2, the required embedding of G is fixed only for con-tree nodes along the con-chain $Q = Q_G(e)$. This requirement is encoded into the C-, P- and R-nodes along Q; for every such node ν we may store its* embedding preference $\pi_e(\nu)$*:*

R-nodes: *The skeleton S_ν of an R-node ν allows only a unique planar embedding and its mirror. We declare one of these two embeddings as the default one. The insertion algorithm then either sets $\pi_e(\nu) :=$* DEFAULT *or $\pi_e(\nu) :=$* MIRROR*, depending on which embedding it requires.*

P-nodes: *The skeleton of a P-node ν with p edges allows $(p-1)!$ planar embeddings given by different cyclic orderings of its edges around one of its nodes. When the con-chain Q passes through a P-node such that one of its neighbors is a C-node, the order of the edges is irrelevant, denoted by $\pi_e(\nu) =$* IRRELEVANT*. Otherwise, the insertion path leaves one of the virtual edges of the skeleton S_ν and enters another one. Hence, these two edges should be neighbors in cyclic order, and the embedding preference is stored as a pair $\pi_e(\nu) := (e_1, e_2)$ which means the skeleton edge e_1 is to occur clockwise directly before e_2.*

C-nodes: *Let $B_1, B_2 \subseteq G$ be the two blocks neighboring a C-node ν on Q. The required embedding places (already embedded) B_2 into a face ϕ_1 of B_1, and vice versa B_1 into a face ϕ_2 of B_2. Unfortunately, those faces ϕ_1, ϕ_2 do not have standalone definitions within G. Let μ_1, μ_2 be the nodes adjacent* *to ν along Q, and let x be the cut vertex of G which is represented by ν. The insertion path within the skeleton S_{μ_i}, $i = 1, 2$, connects x to some other element (i.e., a vertex or a virtual edge) a_i. So, we set the embedding preference to $\pi_e(\nu) := \{a_1, a_2\}$. This means that we will be able to deduce the faces ϕ_1, ϕ_2 (by looking at the canonically computed local insertion subpaths) whenever S_{μ_1}, S_{μ_2} get fixed, and then embed these two faces into each other.*

For all C-, P-, and R-nodes not on the con-chain Q, we do not store any embedding preference. Recall (e.g., from [11, 17]) that S-nodes—representing simple cycles—do not add additional embedding possibilities, and hence the above information is sufficient to determine an embedding of G which allows to insert the edge e with the minimum number of crossings.

When we say we *flip* an embedding of some node $\nu \in V(\mathcal{C})$, then: if ν is an R-node, we switch from $\pi(\nu) = \text{DEFAULT}$ to $\pi(\nu) := \text{MIRROR}$ and vice versa; if ν is a non-IRRELEVANT P-node, we set $\pi(\nu) := (e, f)$ for $\pi(\nu) = (f, e)$; in all other cases, we do not change the encoding. Analogously, the function $\text{FLIPPED}(\pi(\nu))$ gives the so flipped preference of the given preference $\pi(\nu)$.

Observe that a solution of the single edge insertion problem never has a unique embedding—we can always choose the mirror of the identified embedding (i.e, flip all nodes along the corresponding con-chain) and insert the edge analogously, with the same number of crossings.

Before we can discuss how to obtain a fixed embedding $\mathit{\Gamma}$ "close" to the individually preferred embeddings of the edges F, we have to introduce some more formalisms, leading to a lengthy but central definition.

In the following, we will assume some implicit, *fixed orientation* (i.e., direction) of every considered con-chain Q. These orientations are arbitrary and mutually independent; they are needed mainly to consistently speak about *preceding* and *succeeding* nodes on a con-chain, and to break ties in the descriptions.

To simplify notation, we call two nodes μ, ν on a con-chain Q *non-S-neighbors* if none of them is an S-node, and μ, ν are either ordinary neighbors on Q or there is an S-node on Q which is a common neighbor of μ, ν. Let $\pi_{\mathit{\Gamma}}$ be embedding preferences corresponding to a complete embedding $\mathit{\Gamma}$, and π_e preferences from an insertion edge $e \in F$. We say that $\pi_{\mathit{\Gamma}}$ and π_e *agree* on an embedding of a node ν, denoted by $\pi_{\mathit{\Gamma}}(\nu) \simeq \pi_e(\nu)$, iff

- ν is an R- or C-node and $\pi_{\mathit{\Gamma}}(\nu) = \pi_e(\nu)$, or
- ν is a P-node and either $\pi_e(\nu) = \text{IRRELEVANT}$ or the edge pair π_e appears consecutively in clockwise order in $\pi_{\mathit{\Gamma}}$.

They *disagree* ($\not\simeq$) otherwise. We say that a pair of nodes (λ, λ') on a con-chain is *switching* if $\pi_{\mathit{\Gamma}}(\lambda) \simeq \pi_e(\lambda)$ and $\pi_{\mathit{\Gamma}}(\lambda') \simeq \text{FLIPPED}(\pi_e(\lambda'))$, or vice versa. A triple $(\lambda, \lambda', \lambda'')$ is switching if both (λ, λ') and (λ', λ'') are switching.

Definition 3.5 (Dirty pass). *Assume a fixed embedding $\mathit{\Gamma}$ of G, inducing full embedding preferences $\pi_{\mathit{\Gamma}}$ on all nodes of the con-tree $\mathcal{C}(G)$; for P-nodes, let $\pi_{\mathit{\Gamma}}$ give a complete clockwise cyclic ordering of its edges. Let $Q = Q_G(e)$ be the con-chain of an edge $e \in F$, and π_e the corresponding embedding preferences computed in step (2) of the algorithm.*

We say, with respect to the embedding preferences π_Γ (or shortly w.r.t. Γ) and the implicit direction of Q, that a pair $(\mathfrak{c}, e)$ is a dirty pass *(or that $\mathfrak{c}$* is dirty for *e) iff one of the following situations happens:*

i. *$\mathfrak{c} = \nu$ is a C-node on Q such that $\pi_\Gamma(\nu) \not\simeq \pi_e(\nu)$.*
ii. *$\mathfrak{c} = \nu$ is a P-node on Q such that $\pi_e(\nu) \neq$ IRRELEVANT, $\pi_\Gamma(\nu) \not\simeq \pi_e(\nu)$, and $\pi_\Gamma(\nu) \not\simeq$ FLIPPED$(\pi_e(\nu))$.*
iii. *$\mathfrak{c} = (\nu', \nu)$ is a switching pair of R-nodes that are non-S-neighbors (of each other) on Q.*
iv. *$\mathfrak{c} = (\mu', \nu, \mu)$ is a switching triple of nodes such that $\mu \neq \mu'$ are both non-S-neighbors of ν on Q, ν is a P-node, and μ, μ' are P- or R-nodes.*
v. *$\mathfrak{c} = (\mu, \nu)$ is a switching pair of P- or R-nodes that are non-S-neighbors on Q, not both of μ, ν are R-nodes (cf. iii.), and no switching triple as in (iv.) contains both μ, ν.*

X. *We, furthermore, impose the following exclusion rule onto the situations defined above:* No two valid switching triples in (iv.) may share two common nodes. *More precisely, we greedily pack (in the implicit direction on Q) the maximal collection of triples (iv.) on Q, valid according to this exclusion rule.*

Definition 3.5 is essential in the following core claim which shows that dirty passes precisely pinpoint the places where the insertion path(s) need additional crossings.

Lemma 3.6. *Consider a connected graph G with a plane embedding Γ and maximum degree Δ, and an edge e to insert. If there are altogether r_e dirty passes on the con-chain of e w.r.t. Γ, then e can be inserted into Γ with at most* $\mathrm{ins}(G, e) + r_e \cdot \lfloor \Delta/2 \rfloor$ *crossings.*

Proof. Let $Q = Q_G(e)$ be the con-chain of e in $\mathcal{C}(G)$, and denote by Γ_0 an embedding of G which allows to insert e with $\mathrm{ins}(G, e)$ crossings. As discussed in Section 2.2, embedding preferences for nodes not on Q do not matter for e, and hence Γ_0 can be chosen such that it has identical embedding preferences as Γ for all nodes of $\mathcal{C}(G)$ not on Q. Our overall goal is to (locally) modify Γ_0 in the remaining nodes to match given Γ, in a way that creates not many new crossings for e.

For the nodes on Q, we first recall all the mutually exclusive ordered constellations (i.)–(v.) of nodes of Q such that each one of them, according to Definition 3.5, leads to precisely one dirty pass on Q.

We process the nodes λ along Q hierarchically; first considering the subpaths induced by the non-C-nodes in successive order (as implicitly fixed above), and then processing the remaining C-nodes. Our aim is to show the following: If Γ and Γ_0 do not agree on the embedding of the skeleton S_λ, then either we can preserve the insertion path of e without additional crossings, or we identify one of the constellations (i.)–(v.). In the latter case, the insertion path of e can then be rerouted w.r.t Γ's embedding of S_λ with at most $\lfloor \Delta/2 \rfloor$ additional crossings.

For each non-S-node λ to be processed (recall that an S-node has a unique embedding), we denote by λ' the preceding non-S-neighbor of λ on Q; but if we are at the beginning of Q or λ' would be a C-node, then let $\lambda' := \textsc{undefined}$. Let analogously λ'' be the preceding non-S-neighbor of λ' on Q, or $\lambda'' := \textsc{undefined}$.

Let λ be an R-node. Firstly, consider $\pi_\Gamma(\lambda) \simeq \pi_e(\lambda)$. No local change of Γ_0 at λ is necessary. If, moreover, $\pi_\Gamma(\lambda') \simeq \pi_e(\lambda')$ or $\lambda' = \textsc{undefined}$ or $\pi_e(\lambda') = \textsc{irrelevant}$, then also the insertion path of e is locally preserved. If a P-node λ' was as in constellation (ii.) for $\nu = \lambda'$, then the insertion path has been properly adjusted for Γ already at λ' (see below). Otherwise, λ' is a P- or R-node, $\pi_\Gamma(\lambda') \simeq \textsc{flipped}(\pi_e(\lambda'))$, and so (λ', λ) is switching.

Suppose now that λ' is a P-node. Since $\pi_e(\lambda') \neq \textsc{irrelevant}$, also λ'' is defined. In either case—of a switching triple $(\lambda'', \lambda', \lambda)$ as in constellation (iv.) or switching (λ', λ) as in (v.)—we do the following: Let s', t' denote the two vertices of the P-skeleton $S_{\lambda'}$, and $e_{\lambda''}, e_\lambda$ its virtual edges gluing (possibly via intermediate S-nodes) $S_{\lambda''}$ and S_λ to it. The insertion path of e in Γ_0 emanates from $e_{\lambda''}$ straight into e_λ. In Γ, on the other hand, the position of $e_{\lambda''}, e_\lambda$ is flipped (i.e., they are on the "wrong side" of each other). The remedy for Γ is to bring the insertion path close to s' (or t'), and then revolve around s' to the appropriate face of $S_{\lambda'}$ by crossing over at most $\lfloor \Delta/2 \rfloor$ edges incident with s' in G.

Next suppose that λ' is an R-node, i.e. (λ', λ) is as in constellation (iii.). Let $e_{\lambda'} = \{s, t\}$ be the virtual edge of the skeleton S_λ gluing (possibly via an intermediate S-node) $S_{\lambda'}$ to it. As $\pi_\Gamma(\lambda') \simeq \textsc{flipped}(\pi_e(\lambda'))$, the insertion path emanates from $e_{\lambda'}$ in S_λ on the wrong side, but it can be brought to the other side again by revolving around s (or t) at the cost of $\leq \lfloor \Delta/2 \rfloor$ crossings.

Secondly, consider $\pi_\Gamma(\lambda) \simeq \textsc{flipped}(\pi_e(\lambda))$. We alter Γ_0 by replacing the embedding of the skeleton S_λ with its mirror image (while no other skeleton is touched!). Analogically to the first case, if $\lambda' = \textsc{undefined}$, $\pi_e(\lambda') = \textsc{irrelevant}$, or λ' as in (ii.), then the insertion path of e is locally adapted for Γ with no additional crossings. If $\pi_\Gamma(\lambda') \simeq \textsc{flipped}(\pi_e(\lambda'))$, then the relative position of $S_{\lambda'}$ and S_λ remains the same in Γ as it was in Γ_0, and so no extra crossing is needed either.

In the remaining cases when (λ', λ) is switching, the analysis is analogous to the arguments (seen "in a mirror") of the first case, leading again to $\leq \lfloor \Delta/2 \rfloor$ additional crossings in each of the constellations (iii.)–(v.).

Let λ be an P-node. Unless already the same, we modify the embedding by rearranging the order of the virtual edges of the P-skeleton S_λ in Γ_0 to match that of Γ. If $\pi_e(\lambda) = \textsc{irrelevant}$, then no more steps are necessary regarding the insertion path of e at λ. Else, if $\pi_\Gamma(\lambda) \not\simeq \pi_e(\lambda)$ and $\pi_\Gamma(\lambda) \not\simeq \textsc{flipped}(\pi_e(\lambda))$, we are in constellation (ii.), and the insertion path emanating from some virtual edge of S_λ can be rerouted to any other face of S_λ in Γ at the cost of $\leq \lfloor \Delta/2 \rfloor$ crossings in Γ, as above.

In the remaining cases, λ' is a P- or R-node. If (λ', λ) is not switching, then the relative position of $S_{\lambda'}$ and S_λ remains the same in Γ as it was in Γ_0, and so no extra crossing is needed on the insertion path. If λ' is an R-node and (λ', λ) is switching, then we either are in constellation (v.), or constellation (iv.) will occur right in the next step. In the latter case we do nothing for now (as everything will be taken care of next); in the former case we reroute the insertion path at the cost of $\leq \lfloor \Delta/2 \rfloor$ crossings in Γ analogously to the above arguments.

Hence it remains to consider λ' a P-node and (λ', λ) switching. Again, since $\pi_e(\lambda') \neq$ IRRELEVANT, also λ'' is defined. This is the only place where the exclusion principle (X.) for constellations as in (iv.) comes into play: It might happen that neither (v.) with (λ', λ), nor (iv.) with $(\lambda'', \lambda', \lambda)$ (due to (X.)), are valid constellations. In such a case, however, constellation (iv.) will surely occur in the next step, and the insertion path will be properly rerouted then. Otherwise, for constellations (v.) with (λ', λ), or (iv.) with $(\lambda'', \lambda', \lambda)$, we apply the above arguments (since for our rerouting it does not matter whether λ is a P- or an R-node).

Let λ be a C-node. In Γ_0, the face in which the insertion path from one adjacent block B_1 ends, matches the face where the insertion path for the other adjacent block B_2 starts. If constellation (i.) occurs for λ, then those faces may be different in Γ, yet incident to a common cut vertex $x \in B_1 \cap B_2$. Again, we can always locally revolve the insertion path of e around x, thereby crossing at most $\lfloor \Delta/2 \rfloor$ edges.

Since the con-tree $\mathcal{C}(G)$ captures all embedding possibilities for G, the resulting embedding (of the above process) is equivalent to Γ, and the number of crossings on e has risen (since Γ_0) by at most $r_e \cdot \lfloor \Delta/2 \rfloor$. □

Theorem 3.2 now immediately follows from repeated application of Lemma 3.6 to each $e \in F$, and the fact that Algorithm 3.1 computes optimal individual insertion paths within fixed Γ.

3.2 Combining all embedding preferences

We now want to find an embedding Γ that satisfies at least one preference per node of $\mathcal{C}$ and has the property that any pair of con-chains disagrees on at most two dirty passes. For each node in $\mathcal{C}$, we will store a *picked* embedding preference π_{pick}. In the end, these picked embedding preferences will be realized, to subsequently obtain the fixed embedding Γ of Algorithm 3.1, step (3).

Consider a chosen processing order $\langle e_1, e_2, \ldots, e_k \rangle$ of the edges of F: Initially, we set $\pi_{\text{pick}}(\nu) = \emptyset$ for all nodes $\nu \in V(\mathcal{C})$. We consider the edges one by one, setting $\pi_{\text{pick}}(\nu)$ for all nodes ν along the corresponding con-chain of the edge, and probably alter other embedding preferences along the previously considered con-chains (see below for details). After the insertion of the i-th edge, we have the property that each node ν along any con-chain of an edge $e_{i'}$ with $i' \leq i$ has a well-defined $\pi_{\text{pick}}(\nu) \neq \emptyset$.

Let $\mathcal{N}^{<i} \subseteq V(\mathcal{C})$ denote the nodes of $\mathcal{C}$ that have embedding preferences from the first $i-1$ inserted edges ($1 \leq i \leq k+1$). The individual trees within the forest induced by any $\mathcal{N}^{<i}$ give rise to a node partition $\dot{\bigcup}_{j=1}^{\ell} N_j^{<i} = \mathcal{N}$ with $\ell < i$. We call any partition set $N_j^{<i}$ an *embedding part*. Generalizing on the flipping of a single con-chain we can observe:

Proposition 3.7. *Let $N_j^{<i}$ be any embedding part with the embedding preferences π_{pick}. When all the nodes of $N_j^{<i}$ are flipped, we obtain new embedding preferences π'_{pick}. Then, an embedding realizing π'_{pick} allows an insertion of the first $i-1$ insertion edges with the same number of crossings as for π_{pick}. In fact, these insertion paths are identical to the former ones up to mirroring.*

We are now ready to give the following method to obtain a merged embedding Γ. Let π_i be the embedding preferences along the con-chain $Q_i = Q_G(e_i)$ stored in step (2) of Algorithm 3.1.

Algorithm 3.8. Combining all embedding preferences to obtain Γ.

A. Let $\pi_{\text{pick}}(\nu) = \emptyset$, $\forall \nu \in V(\mathcal{C})$.

B. For all $1 \leq i \leq k$:

   (a) Traverse the con-chain Q_i of e_i along some arbitrary, fixed orientation; let $\nu \in Q_i$ be the first node. The traversal direction naturally defines preceding and succeeding nodes along Q_i.

   (b) If $\pi_{\text{pick}}(\nu) = \emptyset$, then choose $\pi_{\text{pick}}(\nu) := \pi_i(\nu)$.

   (c) Let μ be the closest non-S-node preceding ν. Skip this step if μ does not exist or ν is a C-node. Otherwise:
   Check if a flip can improve the embedding: The preference π_{pick} is *improvable* if ν can be the last element of a tuple $\mathfrak{c}$ such that $(\mathfrak{c}, e_i)$ is in a dirty pass w.r.t. Γ, while this tuple would not be dirty after flipping $\pi_{\text{pick}}(\nu)$ (which is equivalent to flipping ν's predecessors).
   If the embedding is improvable, let $Q' \subset V(Q_i)$ be the consecutive nodes of Q_i starting from the start node up to (and including) μ. Furthermore, let N' be all embedding parts of $N^{<i}$ that contain at least one node of Q'. Flip π_{pick} for all nodes $Q' \cup N'$.

   (d) If $\pi_{\text{pick}}(\nu)$ has been newly set in (b), let $\nu' := \nu$. Otherwise let $N_j^{<i}$, for some j, be the embedding part to which ν belonged, and set ν' to the last non-S-node in $N_j^{<i}$ that is traversed by Q.
   Set ν to be the closest non-S-node succeeding ν'; if it exists, continue with step (b), otherwise proceed with the next i in step B.

C. Choose a random embedding preference for any node ν with $\pi_{\text{pick}} = \emptyset$, and randomly complete the embedding preference of any P-node to a complete cyclic ordering. Realize all the preferences π_{pick} to obtain Γ.

Consider the dirty passes that arise from Algorithm 3.8. Thanks to Proposition 3.7 it is easy to see that it is easy to see:

Observation 3.9. *Let π^i_{pick}, $1 \leq i \leq k$, be the picked embedding preference after the insertion of edge e_i in Algorithm 3.8. Let $(\mathfrak{c}, e_j)$, $1 \leq j \leq k$, be a dirty pass w.r.t. the embedding preferences π^ℓ_{pick} of some $\ell \geq j$. Then, the pass is dirty for all the preferences π^m_{pick} with $j \leq m \leq k$.*

the decision whether a pass $(\mathfrak{c}, e_j)$ would be dirty in the final solution is made by the algorithm exactly at the step when merging π_j into the then current π_{pick}. Note that, due to the tree-property of $\mathcal{C}$ (cf. Proposition 2.5), any two con-chains Q, Q' can have at most one common (connected) sub-chain q. The two non-S-nodes closest to either end of q are the two *split nodes* of (Q, Q'). We say a tuple (ν, j, i), $j < i$, is a *splitter* (w.r.t. i) if ν is a split node w.r.t. (Q_j, Q_i). Observe that multiple splitters may induce the split node property of the same node in $\mathcal{C}$. Our key conclusion here reads:

Lemma 3.10. *The above Algorithm 3.8 guarantees that there is at most one dirty pass for each splitter (over all pairs of con-chains); this dirty pass then also contains the corresponding split node. — Hence, the overall sum of dirty passes in the embedding Γ (obtained by Algorithm 3.8) is at most $2\binom{k}{2}$.*

Lemma 3.11 (cf. Lemma 3.10). *The above Algorithm 3.8 guarantees that there is at most one dirty pass for each splitter (over all pairs of con-chains); this dirty pass then also contains the corresponding split node.*

Proof. This can be shown via induction over the following claim, building upon Observation 3.9:

> *Claim.* Let $\pi_o := \pi^{i-1}_{\text{pick}}$ be the intermediate solution of the algorithm after merging the preferences of the first $i-1$ edges. Now, the algorithm computes a new preference $\pi_n := \pi^{i}_{\text{pick}}$ taking the preferences π_i for the con-chain Q_i of the edge e_i into account. Thereby new dirty passes (only ones containing e_i) will arise.
> (a) Each arising dirty pass contains at least one split node w.r.t. some (Q_i, Q_j), $j < i$.
> (b) There is an assignment α_i of splitters w.r.t. i to dirty passes w.r.t. e_i, such that (i) a splitter is only assigned to a pair in which its split node is contained, (ii) each dirty pass has an assigned splitter, and (iii) each splitter is assigned to at most one dirty pass.

This claim trivially holds for $i = 1$. Let $i > 1$, we prove by contradiction.

(a) Assume there would be a dirty pass $(\mathfrak{c}, e_i)$ such that $\mathfrak{c}$ neither is nor contains a split node. Let μ' (μ) be the preceding (succeeding) non-S-neighbor of $\mathfrak{c}$ on Q_i. Since $\mathfrak{c}$ contains no split node, any other con-chain $Q_{j'}$ ($j' < i$) containing $\mathfrak{c}$ also traverses through μ', μ. Let Q_j (if it exists) be the con-chain with the smallest index j, containing $\mathfrak{c}$.

If Q_j does not exist, then all nodes of $\mathfrak{c}$ are set according to π_i. Furthermore, the algorithm performs no flip in step (c) when considering the elements of $\mathfrak{c}$, except probably for the first one. Hence, $\mathfrak{c}$ is not dirty w.r.t. e_i.

Assume Q_j exists. By induction, neither $\mathfrak{c}$ nor subsets thereof are dirty for Q_j. Due to the common neighbors μ', μ, the same routing subproblems for the nodes of $\mathfrak{c}$ are (deterministically) solved for both Q_j, Q_i (but resulting in probably flipped preferences). Hence $\mathfrak{c}$ cannot be a single P- or C-node, but has to be a tuple of two or three nodes. But then, these nodes would not be switching, which is a requirement for any tuple to be part of a dirty pass.

(b) Obtaining such an assignment α_i is trivial for dirty passes $(\mathfrak{c}, e_i)$ where $\mathfrak{c} = \nu$ is a single node, or when at least one node of $\mathfrak{c}$ is a split node only involved in this single dirty pass.

We can hence restrict ourselves to sequences of *incident* node tuples (i.e., tuples having a node in common) along Q_i that are all dirty w.r.t. e_i; the nodes along this sequence that are not contained in two dirty passes are no split nodes.

Recall that no tuples can have more than one node in common, and hence any node is contained in at most two dirty passes. Assume some node ν is involved in two dirty passes but is also contained in at least two splitters; then we could assign one splitter to each of the two passes, and establish α_i for these.

Hence, in order to show that there exists an assignment α_i, it is sufficient to show that there cannot be a sequence D of non-S-neighboring R- and P-nodes, such that the sequence is covered by incident dirty passes, and each split node is contained in two dirty passes but only one splitter.

The algorithm could have avoided any dirty pass along D by flipping (step (c)) when processing the second (or third) node of a any dirty tuple $\mathfrak{c}$. Since no such flips were performed, there have to have been other con-chains before considering Q_i which already put the nodes into a common embedding part.

For notational simplicity, let $\langle \nu_2, \nu_3, ..., \nu_{d-1} \rangle$ denote the nodes of D, in traversal order, that are contained in two dirty passes. Let ν_1 and ν_d are the first and last node of D, respectively (both are no split nodes). If $\nu_{i'}, \nu_{i'+1}$ belong to the same dirty pass defined on a P-node triple, let $\nu_{i'.5}$ denote the center node (which is no split node). Then, let $Q'_{i'}$, $1 \leq i < d$, be the con-chain putting $\nu_{i'}$ and $\nu_{i'+1}$ (and $\nu_{i'.5}$, if it exists) into a common embedding part for the first time during the algorithm. (Note that two such con-chains $Q'_{i'}, Q'_{i''}$ are not necessarily distinct for $i' \neq i''$.) We will inductively (for increasing i') show that $Q'_{i'}$ contains all nodes $\nu_1, \ldots, \nu_{i'+1}$, and that $\nu_{i'+1}$ is a split node w.r.t. $(Q'_{i'}, Q_i)$.

Base case. Consider Q'_1, the con-chain first putting ν_1 and ν_2 into a common embedding part. Since ν_1 is not a split node (and neither is $\nu_{1.5}$ if it exists), Q'_1 also includes ν_1's non-S-neighbor ν_0 preceding D. If Q'_1 would not be the con-chain establishing that ν_2 is a split node, then Q'_1 would also have to include ν_3 (and $\nu_{2.5}$ if it exists). But then, the subproblems at the skeletons of ν_1, ($\nu_{1.5}$,) ν_2 would have been identical for Q'_1 and Q_i. As Q'_1 established their embedding preferences, these nodes would not be in a common dirty pass w.r.t. e_i.

Induction. Consider $Q'_{i'}$, be the con-chain first putting $\nu_{i'}$ and $\nu_{i'+1}$ into a common embedding part. Since $\nu_{i''}$, $1 \leq i'' \leq i'$ cannot be split nodes w.r.t. $(Q'_{i'}, Q_i)$ by induction, they are all contained in $Q'_{i'}$. If $Q'_{i'}$ would not be the con-chain establishing that $\nu_{i'+1}$ is a split node, then $Q'_{i'}$ would also have to include $\nu_{i'+2}$. But then, the subproblems at the skeletons of $\nu_{i'}$, ($\nu_{i'.5}$,) $\nu_{i'+1}$ would have been identical for $Q'_{i'}$ and Q_i. As $Q'_{i'}$ established their embedding preferences, these nodes would not be in a common dirty pass w.r.t. e_i.

Contradiction. Applying the induction for $i' = d-1$ would require ν_d to be a split node, which is a contradiction.

Having shown that the central claim holds, the lemma's result follows from trivial induction. □

As we have already established, any two con-chains give rise to at most 2 splitters, and hence we can conclude:

Corollary 3.12 (cf. Lemma 3.10). *The overall sum of dirty passes in the embedding Γ (obtained by Algorithm 3.8) is at most $2\binom{k}{2}$.*

3.3 Runtime analysis of Algorithm 3.1

As mentioned in Section 2, we can build the con-tree in linear time $O(|V|)$, based on the linear-time decomposition algorithm [21]. In step (2), we call the $O(|V|)$ insertion algorithm k times. Later, we discuss how to implement the merge algorithm (Algorithm 3.8, called in step (3) of the main algorithm) so that it takes at most $O(k|V|)$ time. In step (4), we then run k BFS algorithms, requiring $O(|V|)$ time each. By using suitable tie-breaking, step (5)a) will not be necessary, and since each edge has at most $O(|V|+k)$ crossings in the end, the realization may require up to $O(k|V(G)|+k^2)$ time, which therefore also constitutes the overall runtime bound of the algorithm.

Runtime of Algorithm 3.8. First, observe that checking improvability in step (c) requires only constant time, as the number of cases when a node (node tuple) becomes dirty is constant (and identified P-node triples can be marked, to follow the definition's exclusion rule).

Yet, a naïve implementation would not result in the aforementioned running time as the number of described node flips can easily exceed this bound. Therefore, add two bits to each node in the con-tree: one stores whether a node's embedding preference has to be considered *flipped*, the other one is called *flipmark*. For each iteration of the main loop (i.e., for each edge e_i) these bits are set to *false*. Whenever we should flip the nodes $Q' \cup N'$, we only set the *flipmark* at the node μ instead. At the end of the iteration i, we perform a BFS starting from the last node of Q_i (say ν^*), only considering nodes of Q_i and $N^{<i}$. During this BFS, we flip all visited nodes (by flipping the *flipped* bit), if there has been an odd number of set *flipmark* bits along the path from ν^*. Hence, this operation only requires $O(|V|)$ many operations per iteration i. Realizing all embeddings at step C can then be done in $O(|V|)$ time. Hence, Algorithm 3.8 requires at most $O(k|V|)$ time.

4 Crossing Number Approximations

Our main concept of interest is the crossing number of the graph $G+F$. We can combine our above result with a result of [8], connecting the optimal crossing number with the problem of multiple edge insertion.

Theorem 4.1 (Chimani et al. [8]). *Consider a planar graph G and an edge set F, $F \cap E(G) = \emptyset$. The value* $\mathrm{ins}(G,F)$ *of an optimal solution to* MEI(G,F) *satisfies*

$$\mathrm{ins}(G,F) \;\leq\; 2|F| \cdot \left\lfloor \frac{\Delta(G)}{2} \right\rfloor \cdot \mathrm{cr}(G+F) + \binom{|F|}{2}$$

where $\mathrm{cr}(G+F)$ *denotes the (optimal) crossing number of the graph G including the edges F, and $\binom{|F|}{2}$ thereby accounts for crossings between the edges of F.*

Notice that, when considering the crossing number problem of $G + F$, we may assume G to be connected—otherwise we could "shift" some edges of F to G). Let $k = |F|$, $\Delta = \Delta(G)$. Plugging the estimate of Theorem 4.1 into the place of $\mathrm{ins}^{\Sigma}(G, F) \leq \mathrm{ins}(G, F)$ in Theorem 3.3, and realizing that the $\binom{k}{2}$ term in both estimates stands for the same set of crossings, we immediately obtain

$$\begin{aligned}\mathrm{ins}^{\mathrm{Alg}}(G, F) &\leq 2k \cdot \left\lfloor \frac{\Delta}{2} \right\rfloor \cdot \mathrm{cr}(G + F) + \binom{k}{2} + 2 \left\lfloor \frac{\Delta}{2} \right\rfloor \cdot \binom{k}{2} \\ &= \lfloor \Delta/2 \rfloor \cdot 2k \cdot \mathrm{cr}(G + F) + \big(\lfloor \Delta/2 \rfloor + 1/2\big)(k^2 - k).\end{aligned}$$

Hence we can give the outcome of Algorithm 3.1 as an approximate solution to the crossing number problem on $G + F$, proving:

Theorem 4.2 (Theorem 1.2 (3)). *Given a planar graph G and an edge set F, $F \cap E(G) = \emptyset$, Algorithm 3.1 computes, in $O(|F| \cdot |V(G)| + |F|^2)$ time, a solution to the* $\mathrm{cr}(G + F)$ *problem with the following number of crossings*

$$\mathrm{cr}^{\mathrm{aprx}}(G + F) \leq \lfloor \Delta(G)/2 \rfloor \cdot 2|F| \cdot \mathrm{cr}(G + F) + \big(\lfloor \Delta(G)/2 \rfloor + \tfrac{1}{2}\big)\big(|F|^2 - |F|\big).$$

In [18], furthermore, an algorithm is presented to approximate the crossing number of graphs embeddable in any fixed higher orientable surface. This algorithm lists the technical requirement that G has a "sufficiently dense" embedding on the surface. Yet, as noted in [18], a result like Theorem 4.2 allows to drop this requirement: If the embedding density is small, then the removal of the offending small set(s) of edges is sufficient to reduce the graph genus, while the removed edges can be later inserted into an intermediate planar subgraph of the algorithm.

5 A Note on the Planarization Heuristic

The currently practically strongest heuristic [16] for the crossing number problem is the *planarization heuristic* which starts with a maximal planar subgraph of the given non-planar graph, and then iteratively performs single edge insertions. The crossings of such an insertion are then replaced by dummy nodes such that each edge is inserted into a planar graph. Due to its practical superior performance, often giving the optimal solution [5, 14], it was an open question if this approach unknowingly guarantees some approximation ratio.

By investigating our strategy and proofs, it becomes clear that this approach as such cannot directly give an approximation guarantee: by routing an edge (in an R-node) through another virtual edge (representing a subgraph S) and replacing the crossings with dummy nodes, you essentially *fix* (most of) the embedding of S. This fix might result in $O(n)$ embedding restrictions for further edge insertions, without having an edge that requires this embedding. Therefore the number of dirty passes can no longer be bounded by a function in k. Yet, an implementation realizing the planarization heuristic already contains all the ingredients to obtain our approximation; one "only" has to compute all insertion paths and fix an accordingly merged embedding (Algorithm 3.8), before running the fixed-embedding edge insertion subalgorithm for all inserted edges.

6 Conclusions

We have presented a new approximation algorithm for the multi-edge insertion problem which is faster and simpler that the only formerly known one [10], while at the same time giving better bounds; in fact, in contrast to the former multiplicative approximation, it is the first one with an additive bound. Our algorithm directly leads also to improved approximations (even constant ratio ones over a large class of inputs) for the crossing number problem of graphs in which a given set of edges can be removed in order to obtain a planar subgraph, and for graphs that can be embedded on a surface of some fixed genus.

We conclude with an interesting open problem. We know that multi-edge insertion is NP-hard when the number of inserted edges is part of the input, and it is linear time solvable for the special case of inserting a single edge. What is the complexity of optimally inserting a *constant* number of edges?

References

1. S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56:5:1–5:37, April 2009.
2. D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.
3. S. Cabello and B. Mohar. Crossing and weighted crossing number of near planar graphs. In *Proc. GD '08*, volume 5417 of *LNCS*, pages 38–49. Springer, 2008.
4. S. Cabello and B. Mohar. Adding one edge to planar graphs makes crossing number hard. In *Proc. SoCG '10*, pages 68–76. ACM, 2010.
5. M. Chimani. *Computing Crossing Numbers.* PhD thesis, TU Dortmund, Germany, 2008. `http://www.ae.uni-jena.de/alenmedia/dokumente/ComputingCrossingNumbers_PhDthesis_Chimani_pdf.pdf`.
6. M. Chimani, C. Gutwenger, P. Mutzel, and C. Wolf. Inserting a vertex into a planar graph. In *Proc. SODA '09*, pages 375–383, 2009.
7. M. Chimani and P. Hliněný. A tighter insertion-based approximation of the crossing number. Full version. ArXiv, 2011.
8. M. Chimani, P. Hliněný, and P. Mutzel. Approximating the crossing number of apex graphs. submitted. A preliminary version appeared as a poster at GD '08, LNCS 5417, pp. 432–434, 2009.
9. J. Chuzhoy. An algorithm for the graph crossing number problem. In *Proc. STOC '11, to appear*, 2011.
10. J. Chuzhoy, Y. Makarychev, and A. Sidiropoulos. On graph crossing number and edge planarization. In *Proc. SODA '11*, pages 1050–1069. ACM Press, 2011.
11. G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25:956–997, 1996.
12. G. Even, S. Guha, and B. Schieber. Improved approximations of crossings in graph drawings and vlsi layout areas. *SIAM J. Comput.*, 32(1):231–252, 2002.
13. I. Gitler, P. Hliněný, J. Leanos, and G. Salazar. The crossing number of a projective graph is quadratic in the face-width. *Electronic Notes in Discrete Mathematics*, 29:219–223, 2007.
14. C. Gutwenger. *Application of SPQR-Trees in the Planarization Approach for Drawing Graphs.* PhD thesis, TU Dortmund, Germany, 2010.

15. C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In *Proc. GD '00*, volume 1984 of *LNCS*, pages 77–90. Springer, 2001.
16. C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In *Proc. GD '03*, volume 2912 of *LNCS*, pages 13–24. Springer, 2004.
17. C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005.
18. P. Hliněný and M. Chimani. Approximating the crossing number of graphs embeddable in any orientable surface. In *Proc. SODA '10*, pages 918–927, 2010.
19. P. Hliněný and G. Salazar. On the crossing number of almost planar graphs. In *Proc. GD '05*, volume 4372 of *LNCS*, pages 162–173. Springer, 2006.
20. P. Hliněný and G. Salazar. Approximating the crossing number of toroidal graphs. In *Proc. ISAAC '07*, volume 4835 of *LNCS*, pages 148–159. Springer, 2007.
21. J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
22. W. T. Tutte. *Connectivity in graphs*, volume 15 of *Mathematical Expositions*. University of Toronto Press, 1966.
23. I. Vrt'o. Crossing numbers of graphs: A bibliography. `ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf`, 2011.
24. T. Ziegler. *Crossing Minimization in Automatic Graph Drawing*. PhD thesis, Saarland University, Germany, 2001.